



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Mail Stop Appeal Brief-Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, on February 21, 2006

Robert E. Malm

In re Application of:

CUREY et al.

Serial Number: 09/821,537

Filing Date: 03/28/2001

For: PARTITIONED EXECUTIVE
STRUCTURE FOR REAL-TIME
PROGRAMS

Group Art Unit: 2152

Examiner: NABIL EL-HADY

Telephone: (703) 305-8646

**SECOND AMENDMENT TO
REVISED SUPPLEMENTAL APPEAL BRIEF**

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Responding to Notification of Non-Compliant Appeal Brief dated 02/13/2006, applicants request that the accompanying Appeal Brief be substituted for the Appeal Brief previously submitted.

The page numbers have been changed so that there is no discontinuity in the page numbers.

Application No. 09/821,537
Filing Date: 03/28/2001
Inventors: Curey et al.

February 20, 2006

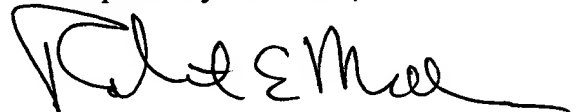
P573C

The page numbers in the Table of Contents correctly reference the sections in the Appeal Brief.

The redundant Section III in the Table of Contents has been deleted.

I believe these submissions remedy the non-compliance issues raised by the examiner.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "Robert E. Malm", with a long horizontal flourish extending to the right.

Robert E. Malm
Reg. No. 34,662

Date: 02/20/2006
Telephone: (310) 573-1781

Enc: Appeal Brief



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, Mail Stop Appeal Brief - Patents, Box 1450, Alexandria, VA 22313-1450 on February 21, 2006.

Robert E. Malm

In re Application of:

RANDALL K. CUREY et al.

Serial Number: 09/821,537

Filing Date: 03/28/2001

For: PARTITIONED EXECUTIVE STRUCTURE
FOR REAL-TIME PROGRAMS

Group Art Unit: 2154

Examiner: NABIL EL-HADY

Telephone: 703 308 7990

REVISED SUPPLEMENTAL APPEAL BRIEF

(REVISED PURSUANT TO REQUIREMENTS OF 37 CFR § 41.37)

(FURTHER REVISED TO CORRECT PAGE NUMBERS)

Submitted by:

Robert E. Malm

Mailing Address:

16624 Pequeno Place
Pacific Palisades, CA 90272

Telephone:

(310) 459-8728



TABLE OF CONTENTS

TABLE OF AUTHORITIES	IV
INTRODUCTION	1
REAL PARTY IN INTEREST	6
RELATED APPEALS AND INTERFERENCES	6
STATUS OF CLAIMS	6
STATUS OF AMENDMENTS	6
SUMMARY OF CLAIMED SUBJECT MATTER	7
GROUND'S OF REJECTION TO BE REVIEWED ON APPEAL	17
ARGUMENT	19
I. WHETHER CLAIMS 1, 25, AND 26 ARE UNPATENTABLE UNDER NON-STATUTORY DOUBLE-PATENTING DOCTRINE IN VIEW OF MARK ET AL. (U.S. 6,829,763)	19
II. WHETHER CLAIMS 1-49 ARE UNPATENTABLE UNDER 35 U.S.C. § 112, SECOND PARAGRAPH, AS BEING INDEFINITE.	20
CLAIMS 1 AND 26	20
CLAIMS 21 AND 46	21
CLAIMS 23 AND 48	22
CLAIMS 24 AND 49	23
CLAIMS 2 AND 27	23
CLAIMS 3 AND 28	24
CLAIMS 6-7 AND 31-32	25
CLAIMS 18 AND 43	26
CLAIMS 22 AND 47	27
III. WHETHER CLAIM 25 IS UNPATENTABLE UNDER 35 U.S.C. § 112, SECOND PARAGRAPH, FOR FAILING TO SET FORTH THE SUBJECT MATTER WHICH APPLICANTS REGARD AS THEIR INVENTION.	30
IV. WHETHER CLAIM 25 IS UNPATENTABLE UNDER 35 U.S.C. § 101 BECAUSE IT EMBRACES OR OVERLAPS TWO DIFFERENT STATUTORY CLASSES	32
V. WHETHER CLAIMS 1, 8, 23, 25, 26, 33, AND 48 ARE UNPATENTABLE UNDER 35 U.S.C. § 102(B) AS BEING ANTICIPATED BY BLUM ET AL. (U.S. 4,109,311)	33
CLAIMS 1, 25, AND 26	33
CLAIMS 8 AND 33	43
CLAIMS 23 AND 48	44
VI. WHETHER CLAIMS 10-11, 13-14, 18, 35-36, 38-39, AND 43 ARE UNPATENTABLE UNDER 35 U.S.C. § 103(A) IN VIEW OF BLUM ET AL. (U.S. 4,109,311)	45
CLAIMS 10 AND 35	45
CLAIMS 11 AND 36	47
CLAIMS 13 AND 38	48
CLAIMS 14 AND 39	50
CLAIMS 18 AND 43	52
CONCLUSIONS	54
CLAIMS APPENDIX	56

EVIDENCE APPENDIX.....	63
COPY OF TERMINAL DISCLAIMER.....	64





TABLE OF AUTHORITIES

CASES

<i>Corning Glass Works v. Sumitomo Elec. U.S.A., Inc.</i> , 868 F.2d 1251, 1257, 9 USPQ2d 1962, 1966 (Fed. Cir. 1989)	35
<i>Ex parte Lyell</i> , 17 USPQ2d 1548 (BPAI 1990)	29
<i>Ex parte Porter</i> , 25 USPQ2d 1144, 1147 (BPAI 1992)	31
<i>In re Donaldson</i> , 16 F.3d 1189, 29 USPQ2d 1845 (Fed. Cir. 1994)	5, 38, 41
<i>In re Lee</i> , 277 F.3d 1338, 1344-45, 61 USPQ2d 1430, 1434-35 (Fed. Cir. 2002)	46, 47, 50
<i>In re Stencel</i> , 828 F.2d 751, 4 USPQ2d 1071 (Fed. Cir. 1987)	35
<i>In re Vaeck</i> , 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991)	44
<i>In re Zurko</i> , 258 F.3d 1379, 1386, 59 USPQ2d 1693, 1697 (Fed. Cir. 2001)	46, 47, 50
<i>Pac-Tec Inc. v. Amerace Corp.</i> , 903 F.2d 796, 801, 14 USPQ2d 1871, 1876 (Fed. Cir. 1990)	35
<i>Pitney Bowes, Inc. v. Hewlett-Packard Co.</i> , 182 F.3d 1298, 1305, 51 USPQ2d 1161, 1165 (Fed. Cir. 1999)	36
<i>Richardson v. Suzuki Motor Co.</i> , 868 F.2d 1226 9 USPQ2d 1913, 1920 (Fed. Cir. 1989)	32
<i>Verdegaal Bros. v. UnionOil Co. of California</i> , 814 F.2d 628, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987)	32

STATUTES

35 U.S.C. § 101	31
35 U.S.C. § 102(b)	32
35 U.S.C. § 103(a)	44
35 U.S.C. § 112, 2nd paragraph	19, 30
35 U.S.C. § 112, 6th paragraph	5

OTHER AUTHORITIES

<i>Encyclopedia of Computer Science</i>	21, 23, 45, 48, 50
<i>McGraw-Hill Dictionary of Scientific and Technical Terms</i>	25, 51
MPEP § 2111.01	5
MPEP § 2111.02	35, 36
MPEP § 2131	32
MPEP § 2142	44
MPEP § 2144.03 A	47, 49
MPEP § 2144.03 B	46, 48, 49, 50
MPEP § 2173.05(e)	26, 28
MPEP § 2173.05(p)	29
MPEP § 2181 I	38, 41
MPEP § 806.05(e)	31
<i>Random House College Dictionary</i>	24, 42



INTRODUCTION

The invention that is the subject matter of this appeal is a method and apparatus for executing a plurality of software packages at a plurality of rates utilizing a common set of computational resources. The basis of the invention is to assign for execution each of the plurality of software packages to a particular sequence of time intervals, the sequence of time intervals assigned to one software package not overlapping the sequence of time intervals assigned to any other software package.

The invention can be analogized to a method for controlling access to a computer in a library. The librarian maintains a list of library users who wish to use the computer and assigns a sequence of regularly-occurring half-hour time intervals to each user in accordance with the user's needs. One user might be assigned the 800-830 segment each day. Another user might be assigned the 1030-1100 segment each Monday. Still another user might be assigned the 1400-1430 segment every fourth Wednesday. The librarian enters these assignments on a calendar which is posted on the bulletin board. Users take control of the computer in accordance with their assigned time intervals and the time, including day and month, as shown by the clock on the wall.

The librarian (L) applies for a patent for her method of controlling access to the computer. L's application contains a single claim:

A method for controlling access to a computer whereby each computer user has access to the computer during each time interval of a sequence of time intervals assigned to the user.

The examiner rejects L's claim on the basis of M's patent which discloses a method whereby the computer supervisor keeps a list of users who wish to periodically use the computer. When a user takes control of the computer, the computer supervisor starts a half-hour interval timer and

posts the name of the next user on the list. When the half-hour period expires, the user whose name is posted takes control of the computer.

Are L's method and M's method the same? The examiner concluded that they were by observing that over time, a user using M's method had access to the computer during each time interval of a sequence of time intervals.

L argued that the examiner was ignoring the words "assigned to the user" when he compared M's invention with L's claim. A sequence of time intervals is never assigned to a user in M's invention. Instead, only the next-occurring time interval is ever assigned. A user is assigned a time interval when the user's name is the next name on the list of computer users after the name of the user currently using the computer. And this assignment occurs when the librarian posts the name of the user.

We return now to the subject matter of this appeal. This appeal has to do with a method and apparatus for executing a plurality of software packages at one or more rates utilizing a common set of computational resources. The basis of the invention is to generate non-overlapping sequences of time intervals and then to assign for execution each of the plurality of software packages to a particular sequence of time intervals.

Claim 1 of appellants' application claims a method for executing a plurality of software packages and reads as follows.

1. *A method [1] for repetitively executing a plurality of software packages at one or more rates, utilizing a common set of computational resources, the method comprising the steps:*

[2] generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages;

[3] executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals.

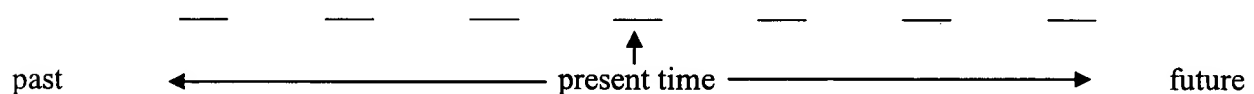
The numbered limitations in boldface are not disclosed by the prior art.

The examiner cited Blum et al. in rejecting this claim and apparatus claims 25 and 26 which contain essentially the same limitations.

The Blum et al. invention is a way of having a single processor execute a plurality of programs simultaneously by interleaving the execution of the programs in time. The executing microprocessor maintains a list of programs to be executed. A single listing of a program is allowed to execute for no more than one "time slice", the duration of a time slice being fixed for all programs.

The listed programs are executed, a few instructions at a time, by repetitively cycling through the list of programs. (A program may be listed a number of times if it is desired to have more time for the program to execute during a single cycle through the list.)

Limitation [2] of claim 1 calls for ***generating a sequence of time intervals for each of the plurality of software packages***. If a mathematician were to generate a sequence of time intervals, he could represent it by a sequence of solid line segments as follows:



One might generate this sequence electronically by a circuit which produced a square wave with the high levels corresponding to the solid line segments above. In accordance with limitation [2], we assign this sequence of time intervals to Software Package No. 1.

Limitation [3] of claim 1 calls for *executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals*. In the case of Software Package No. 1 and the assignment of the sequence of time intervals shown above, Software Package No. 1 would be executed each time the square wave was high.

Let us now compare Blum et al.'s execution process with appellants' invention as defined by claim-1's limitations [2] and [3]. Blum et al. assigns the next occurring time slice to the program following the currently-executing program on the list of programs to be executed. Blum et al. generates a sequence of pulses as shown below for timing purposes:



The pulses are denoted by the vertical lines. Each pulse represents the end of one time slice and the beginning of the next. The occurrence of a pulse signals the end of the time period allotted to the currently-executing program and the beginning of the time period allotted to the next program on the list. This sequence of pulses is not an assignment of execution time intervals to any particular software package, and Blum et al. does not disclose the generation of any other sequences of time intervals.

There is simply no disclosure in Blum et al. of *generating a sequence of time intervals for each of the plurality of software packages*.

It follows that there can be no disclosure of Limitation [3] since there can be no execution of each software package during the time intervals of its sequence of time intervals when no sequences of time intervals have been assigned to the software packages.

The examiner's "interpretation" of the claim-1 limitations are considerably different than appellants'. Appellants believe that the claims are sufficiently unambiguous as to preclude

interpretations such as the examiner's. However, recognizing that differences of opinion are always possible, appellants respectfully request the Board to resolve any questions concerning claim interpretation by consulting appellants' specification and drawings in accordance with *In re Donaldson*:

"During examination, the claims must be interpreted as broadly as their terms reasonably allow. . . . There is one exception and that is when an element is claimed using language falling under the scope of 35 U.S.C. 112, 6th paragraph (often broadly referred to as means or step plus function language). In that case the specification must be consulted to determine the structure, material, or acts corresponding to the function recited in the claim. *In re Donaldson*, 29 USPQ2d 1845 (Fed. Cir. 1994)(see MPEP ¶ 2181 - ¶ 2186)." Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2111.01.

The limitations of the method claims are all step-plus-function limitations (i.e. the steps do not specify acts sufficient to accomplish the specified functions) and the limitations of the apparatus claims are all means-plus-function limitations (i.e. the means do not specify structure sufficient to accomplish the specified functions). The claims are clearly within the scope of 35 U.S.C. § 112, 6th paragraph.

REAL PARTY IN INTEREST

The real party in interest is: NORTHROP GRUMMAN CORPORATION
1840 Century Park East, 18th Floor
Los Angeles, CA 90067-2199

RELATED APPEALS AND INTERFERENCES

The application is a CIP of Application 09/572,298, filed 05/16/2000, which was appealed from the examiner to the Board of Patent Appeals and Interferences, then withdrawn from appeal by the patent office, and allowed to issue on December 7, 2004 as Patent No. 6,829,763.

STATUS OF CLAIMS

Claims 1-49 are pending in the application.

Claims 1-49 were rejected and all are subject to appeal.

STATUS OF AMENDMENTS

No amendments were requested subsequent to the examiner's final rejection of the claims.

SUMMARY OF CLAIMED SUBJECT MATTER

SUMMARY OF CLAIMED SUBJECT MATTER FOR INDEPENDENT CLAIM 1

The invention is a method for repetitively executing a plurality of software packages at a plurality of rates utilizing a common set of computational resources. The method consists of counting contiguous time increments and executing a plurality of software packages. Specification: p. 4, lines 12-21. Each software package is executed during each time increment in one or more sequences of time increments. The time increments in each sequence recur at a predetermined rate, and the time increments assigned to one software package do not overlap the time increments assigned to any other of the plurality of software packages. Specification: p. 4, lines 5-11; p. 10, lines 18-22.

SUMMARY OF CLAIMED SUBJECT MATTER FOR INDEPENDENT CLAIM 26

The invention is an apparatus for repetitively executing a plurality of software packages at a plurality of rates utilizing a common set of computational resources. The apparatus counts contiguous time increments and executes a plurality of software packages. Specification: p. 4, lines 12-21. The apparatus executes each software package during each time increment in one or more sequences of time increments. The time increments in each sequence recur at a predetermined rate, and the time increments assigned to one software package do not overlap the time increments assigned to any other of the plurality of software packages. Specification: p. 4, lines 5-11; p. 10, lines 18-22.

MEANS-PLUS-FUNCTIONS & STEP-PLUS-FUNCTIONS

1. A method for repetitively executing a plurality of software packages at one or more rates, utilizing a common set of computational resources, the method comprising the steps:

[step plus function] generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages (Specification: p. 4, lines 5-11; p. 10, lines 18-22);

[step plus function] executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals (Specification: p. 4, lines 12-21).

2. The method of claim 1 **[further limitation of step plus function]** wherein the plurality of software packages of the "executing" step includes only valid software packages (Specification: p. 11, lines 7-9), the method further comprising the step:

[step plus function] utilizing one or more tests to identify the software packages that are valid (Specification: p. 11, lines 9-10).

3. The method of claim 2 **[further limitation of step plus function]** wherein one of the tests for validity is a one's complement checksum test of a software package's program memory (Specification: p. 11, lines 10-11).

4. The method of claim 2 **[further limitation of step plus function]** wherein a software package is assigned its own dedicated memory region, one of the tests for validity being whether the address returned for the software package's initialization procedure lies within its dedicated memory region (Specification: p. 11, lines 11-12).

5. The method of claim 4 **[further limitation of step plus function]** wherein one of the tests is whether the address is returned within a predetermined time (Specification: p. 11, lines 15-17).

6. The method of claim 2 **[further limitation of step plus function]** wherein a software package is assigned its own dedicated memory region, the software package's dedicated memory region including a stack memory region and/or a heap memory region (Specification: p. 8, lines 5-7), one of the tests for validity being whether the stack memory range and/or the heap memory range assigned during the execution of the software package's initialization procedure and the various associated entry points lies within the software package's dedicated memory region (Specification: p. 11, lines 12-15).

7. The method of claim 6 **[further limitation of step plus function]** wherein one of the tests is whether the stack memory range and/or the heap memory range and the various associated entry points are returned within a predetermined time (Specification: p. 11, lines 15-17).

8. The method of claim 1 **[further limitation of step plus function]** wherein a software package is assigned its own dedicated memory region (Specification: p. 8, lines 5-7).

9. The method of claim 8 **[further limitation of step plus function]** wherein the software package's dedicated memory region includes a stack memory region, a software package's stack residing in the software package's stack memory region (Specification: p. 8, lines 5-7).

10. The method of claim 1 **[further limitation of step plus function]** wherein a software package includes background tasks as well as foreground tasks, the background tasks being performed after the foreground tasks have been completed (Specification: p. 12, lines 8-14).

11. The method of claim 10 **[further limitation of step plus function]** wherein a background task is an infinite loop (Specification: p. 12, lines 10-12).

12. The method of claim 10 **[further limitation of step plus function]** wherein the software package causes the power utilized in executing the software package to be minimized after completion of the background tasks (Specification: p. 12, lines 12-14).

13. The method of claim 1 **[further limitation of step plus function]** wherein a failure in the execution of a software package causes information to be logged in a failure log (Specification: p. 12, lines 15-18).

14. The method of claim 13 **[further limitation of step plus function]** wherein a failure in execution is linked to the software package that caused the failure (Specification: p. 12, lines 15-16).

15. The method of claim 13 **[further limitation of step plus function]** wherein quality of performance in executing a software package is represented by one or more performance-quality parameters, values of the one or more performance-quality parameters being determined from the information logged in a failure log, the execution of a software package being subject to a plurality of execution options, an execution option being selected on the basis of one or more performance-quality parameter values (Specification: p. 12, lines 15-22).

16. The method of claim 15 **[further limitation of step plus function]** wherein the plurality of execution options are user configurable (Specification: p. 12, lines 21-22).

17. The method of claim 15 **[further limitation of step plus function]** wherein performance-quality parameters include the number of failures and/or the rate of failures for one or more classes of failures recorded in a software package's failure log (Specification: p. 12, lines 19-20).

18. The method of claim 1 **[further limitation of step plus function]** wherein safety-critical software is placed in one or more separate partitions thereby isolating the safety-critical software from non-safety-critical software (Specification: p. 13, lines 1-2).

19. The method of claim 1 **[further limitation of step plus function]** wherein each of the plurality of software packages is assigned its own memory block, a software package being enableable to read data only from zero or more memory blocks associated with other software packages, the zero or more memory blocks readable by a software package being either predetermined or determined during execution of the software packages in accordance with a set of one or more rules (Specification: p. 8, line 20 - p. 9, line 14).

20. The method of claim 1 **[further limitation of step plus function]** wherein each of the plurality of software packages is assigned its own memory block, a software package being enableable to write data only to zero or more memory blocks associated with other software packages, the zero or more memory blocks writeable by a software package being either predetermined or determined during execution of the software packages in accordance with a set of one or more rules (Specification: p. 8, lines 20 - p. 9, line 6 and lines 15-18).

21. The method of claim 1 **[further limitation of step plus function]** wherein an executive software package enforces the discipline that each software package executes only during the time intervals of its sequence of time intervals, the executive software package determining when the execution of a software package extends into a time interval belonging to the sequence of time intervals assigned to another software package and performs a remedial action (Specification: p. 10, line 18 - p. 11, line 6).

22. The method of claim 1 **[further limitation of step plus function]** wherein the presence of those software packages that are present is detected (Specification: p. 11, lines 7-18).

23. The method of claim 1 **[further limitation of step plus function]** wherein one or more of the plurality of software packages are independently compiled, linked, and loaded (Specification: p. 10, lines 7-8).

24. The method of claim 1 **[further limitation of step plus function]** wherein a software package has its own stack, the software package's stack being selected prior to executing the software package (Specification: p. 11, lines 19-22).

25. **[means plus function]** Apparatus for practicing the method of claim 1 (Specification: p. 4, lines 5-21; p. 10, lines 18-22).

26. Apparatus for repetitively executing a plurality of software packages at a plurality of rates, the apparatus comprising:

[means plus function] a means for generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages (Specification: p. 4, lines 5-11; p. 10, lines 18-22);

[means plus function] a means for executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals (Specification: p. 4, lines 12-21).

27. The apparatus of claim 26 **[further limitation of means plus function]** wherein the plurality of software packages executed by the "executing" means includes only valid software packages (Specification: p. 11, lines 7-9), the apparatus further comprising:

[means plus function] a means for utilizing one or more tests to identify the software packages that are valid (Specification: p. 11, lines 9-10).

28. The apparatus of claim 27 **[further limitation of means plus function]** wherein one of the tests for validity is a one's complement checksum test of a software package's program memory (Specification: p. 11, lines 10-11).

29. The apparatus of claim 27 **[further limitation of means plus function]** wherein a software package is assigned its own dedicated memory region, one of the tests for validity being whether the address returned for the software package's initialization procedure lies within its dedicated memory region (Specification: p. 11, lines 11-12).

30. The apparatus of claim 29 **[further limitation of means plus function]** wherein one of the tests is whether the address is returned within a predetermined time (Specification: p. 11, lines 15-17).

31. The apparatus of claim 27 **[further limitation of means plus function]** wherein a software package is assigned its own dedicated memory region, the software package's dedicated memory region including a stack memory region and/or a heap memory region (Specification: p. 8, lines 5-7), one of the tests for validity being whether the stack memory range and/or the heap memory range assigned during the execution of the software package's initialization procedure and the various associated entry points lies within the software package's dedicated memory region (Specification: p. 11, lines 12-15).

32. The apparatus of claim 31 **[further limitation of means plus function]** wherein one of the tests is whether the stack memory range and/or the heap memory range and the various associated entry points are returned within a predetermined time (Specification: p. 11, lines 15-17).

33. The apparatus of claim 26 **[further limitation of means plus function]** wherein a software package is assigned its own dedicated memory region (Specification: p. 8, lines 5-7).

34. The apparatus of claim 33 **[further limitation of means plus function]** wherein the software package's dedicated memory region includes a stack memory region, a software package's stack residing in the software package's stack memory region (Specification: p. 8, lines 5-7).

35. The apparatus of claim 26 **[further limitation of means plus function]** wherein a software package includes background tasks as well as foreground tasks, the background tasks being performed after the foreground tasks have been completed (Specification: p. 12, lines 8-14).

36. The apparatus of claim 35 **[further limitation of means plus function]** wherein a background task is an infinite loop (Specification: p. 12, lines 10-12).

37. The apparatus of claim 35 **[further limitation of means plus function]** wherein the software package causes the power utilized in executing the software package to be minimized after completion of the background tasks (Specification: p. 12, lines 12-14).

38. The apparatus of claim 26 **[further limitation of means plus function]** wherein a failure in the execution of a software package causes information to be logged in a failure log (Specification: p. 12, lines 15-18).

39. The apparatus of claim 38 **[further limitation of means plus function]** wherein a failure in execution is linked to the software package that caused the failure (Specification: p. 12, lines 15-16).

40. The apparatus of claim 38 **[further limitation of means plus function]** wherein quality of performance in executing a software package is represented by one or more performance-quality parameters, values of the one or more performance-quality parameters being determined from the information logged in a failure log, the execution of a software package being subject to a plurality of execution options, an execution option being selected on the basis of one or more performance-quality parameter values (Specification: p. 12, lines 15-22).

41. The apparatus of claim 40 **[further limitation of means plus function]** wherein the plurality of execution options are user configurable (Specification: p. 12, lines 21-22).

42. The apparatus of claim 40 **[further limitation of means plus function]** wherein performance-quality parameters include the number of failures and/or the rate of failures for one or more classes of failures recorded in a software package's failure log (Specification: p. 12, lines 19-20).

43. The apparatus of claim 26 **[further limitation of means plus function]** wherein safety-critical software is placed in one or more separate partitions thereby isolating the safety-critical software from non-safety-critical software (Specification: p. 13, lines 1-2).

44. The apparatus of claim 26 **[further limitation of means plus function]** wherein each of the plurality of software packages is assigned its own memory block, a software package being enableable to read data only from zero or more memory blocks associated with other software packages, the zero or more memory blocks readable by a software package being either predetermined or determined during execution of the software packages in accordance with a set of one or more rules (Specification: p. 8, line 20 - p. 9, line 14).

45. The apparatus of claim 26 **[further limitation of means plus function]** wherein each of the plurality of software packages is assigned its own memory block, a software package being enableable to write data only to zero or more memory blocks associated with other software packages, the zero or more memory blocks writeable by a software package being either predetermined or determined during execution of the software packages in accordance with a set of one or more rules (Specification: p. 8, line 20 - p. 9, line 6 and lines 15-18).

46. The apparatus of claim 26 **[further limitation of means plus function]** wherein an executive software package enforces the discipline that each software package executes only during

the time intervals of its sequence of time intervals, the executive software package determining when the execution of a software package extends into a time interval belonging to the sequence of time intervals assigned to another software package and performs a remedial action (Specification: p. 10, line 18 - p. 11, line 6).

47. The apparatus of claim 26 **[further limitation of means plus function]** wherein the presence of those software packages that are present is detected (Specification: p. 11, lines 7-18).

48. The apparatus of claim 26 **[further limitation of means plus function]** wherein one or more of the plurality of software packages are independently compiled, linked, and loaded (Specification: p. 10, lines 7-8).

49. The apparatus of claim 26 **[further limitation of means plus function]** wherein a software package has its own stack, the software package's stack being selected prior to executing the software package (Specification: p. 11, lines 19-22).

GROUND OF REJECTION TO BE REVIEWED ON APPEAL

I. WHETHER CLAIMS 1, 25, AND 26 ARE UNPATENTABLE UNDER NON-STATUTORY DOUBLE-PATENTING DOCTRINE IN VIEW OF MARK ET AL. (U.S. 6,829,763).

II. WHETHER CLAIMS 1-49 ARE UNPATENTABLE UNDER 35 U.S.C. § 112, SECOND PARAGRAPH, AS BEING INDEFINITE.

III. WHETHER CLAIM 25 IS UNPATENTABLE UNDER 35 U.S.C. § 112, SECOND PARAGRAPH, FOR FAILING TO SET FORTH THE SUBJECT MATTER WHICH APPLICANTS REGARD AS THEIR INVENTION.

IV. WHETHER CLAIM 25 IS UNPATENTABLE UNDER 35 U.S.C. § 101 BECAUSE IT EMBRACES OR OVERLAPS TWO DIFFERENT STATUTORY CLASSES.

V. WHETHER CLAIMS 1, 8, 23, 25, 26, 33, AND 48 ARE UNPATENTABLE UNDER 35 U.S.C. § 102(B) AS BEING ANTICIPATED BY BLUM ET AL. (U.S. 4,109,311).

VI. WHETHER CLAIMS 10-11, 13-14, 18, 35-36, 38-39, AND 43 ARE UNPATENTABLE UNDER 35 U.S.C. § 103(A) IN VIEW OF BLUM ET AL. (U.S. 4,109,311).

GROUPING OF CLAIMS

Insofar as Ground I is concerned:

Claims 1, 25, and 26 stand or fall together;

Insofar as Ground II is concerned:

Claims 1 and 26 stand or fall together;

Claims 21 and 46 stand or fall together;

Claims 23 and 48 stand or fall together;

Claims 24 and 49 stand or fall together;

Claims 2 and 27 stand or fall together;

Claims 3 and 28 stand or fall together;

Claims 6 and 31 stand or fall together and claims 7 and 32 stand or fall together;

Claims 18 and 43 stand or fall together;

Claims 22 and 47 stand or fall together;

All other claims stand or fall with the claims from which they depend.

Insofar as Ground III is concerned:

Claim 25 stands or falls by itself.

Insofar as Ground IV is concerned:

Claim 25 stands or falls by itself.

Insofar as Ground V is concerned:

Each of claims 1, 8, 23, 25, 26, 33, and 48 stands or falls by itself.

Insofar as Ground VI is concerned:

Each of claims 10-11, 13-14, 18, 35-36, 38-39, and 43 stands or falls by itself.

ARGUMENT

I. WHETHER CLAIMS 1, 25, AND 26 ARE UNPATENTABLE UNDER NON-STATUTORY DOUBLE-PATENTING DOCTRINE IN VIEW OF MARK ET AL. (U.S. 6,829,763).

The present application and U.S. Patent 6,829,763 are both owned by Northrop Grumman Corporation. A terminal disclaimer in compliance with 37 CFR 1.321c has been filed with the U.S. Patent & Trademark Office for the purpose of obviating the double-patenting rejection by the examiner.

A copy of the terminal disclaimer is attached to this brief.

II. WHETHER CLAIMS 1-49 ARE UNPATENTABLE UNDER 35 U.S.C. § 112, SECOND PARAGRAPH, AS BEING INDEFINITE.

CLAIMS 1 AND 26

Claims 1 and 26 read as follows:

1. A method for repetitively executing a plurality of software packages at one or more rates, utilizing a common set of computational resources, the method comprising the steps:
generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages;
executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals.

26. Apparatus for repetitively executing a plurality of software packages at a plurality of rates, the apparatus comprising:
a means for generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages;
a means for executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals.

Appellants specify in the claim preambles a method (claim 1) and apparatus (claim 26) for repetitively executing "a plurality of software packages". This recitation provides the antecedent basis for "THE plurality of software packages" which appears in the first element of both claims.

"A plurality of software packages" that appears in the second element of both claims, in accordance with common practice, denotes a different set of software packages. (It does not use the phrase "the plurality of software packages".) From the context (i.e. "each software package being executed during the time intervals of its sequence of time intervals"), we know that this different set of software packages is made up of one or more of the "plurality of software packages" referred to in the preamble and the first element.

The examiner argues: "However, if "a plurality of software packages" does mean a subset of the previously defined "the plurality of software packages", or if it does mean a set of control software packages different from "the plurality of software packages", then it is not clearly understood on what bases, how or why, this subset of this set is being chosen/generated to be executed." 10/22/04 Office Action, pp. 3-4, § 6.

The "plurality of software packages" referred to in the second element of the claims has to be a subset of the "plurality of software packages" referred to in the preamble since it is only these software packages that have assigned sequences of time intervals. With regard to how "plurality of software packages" referred to in the second element of the claims are chosen, the user of the invention may choose them in any way he so desires. There is no requirement that a basis for the choice be stated in the claims.

CLAIMS 21 AND 46

Claims 21 and 46 read as follows:

21. The method of claim 1 wherein an executive software package enforces the discipline that each software package executes only during the time intervals of its sequence of time intervals, the executive software package determining when the execution of a software package extends into a time interval belonging to the sequence of time intervals assigned to another software package and performs a remedial action.

46. The apparatus of claim 26 wherein an executive software package enforces the discipline that each software package executes only during the time intervals of its sequence of time intervals, the executive software package determining when the execution of a software package extends into a time interval belonging to the sequence of time intervals assigned to another software package and performs a remedial action.

The examiner argues that it is not clear as to which plurality of software packages is to be associated with "each software package", "a software package", and "another software package".

10/22/04 Office Action, p. 4, § 6.

Since claims 21 and 46 are limitations pertaining to the second element of claims 1 and 26 (i.e. they deal with execution of the software packages), it is clear that they must be associated with the "plurality of software packages" that appears in the second element of both claims.

CLAIMS 23 AND 48

Claims 23 and 48 read as follows:

23. The method of claim 1 wherein one or more of the plurality of software packages are independently compiled, linked, and loaded.

48. The apparatus of claim 26 wherein one or more of the plurality of software packages are independently compiled, linked, and loaded.

The "load" operation is performed by a "loader", the function of which is "to transfer a program held on some external storage medium such as magnetic disk into the main memory of the machine in a form suitable for execution." *Encyclopedia of Computer Science*, Third Edition, Ralston & Reilly, editors, Van Nostrand Reinhold, New York, N.Y., 1993. Claims 23 and 48 are obviously limitations of the second element of claims 1 and 26 respectively having to do with executing programs. It follows that the antecedent basis of "the plurality of software packages" referred to in the claims is the phrase "a plurality of software packages" appearing in the second element of the claims.

The examiner argues that the claims are unclear (i.e. which "plurality of software packages" is being referred to). 10/22/04 Office Action, p. 4, § 6. However, a person skilled in the art of computer systems would have no difficulty in understanding the meanings of the claims by consulting an appropriate encyclopedia if necessary.

CLAIMS 24 AND 49

Claims 24 and 49 read as follows:

24. The method of claim 1 wherein a software package has its own stack, the software package's stack being selected prior to executing the software package.

49. The apparatus of claim 26 wherein a software package has its own stack, the software package's stack being selected prior to executing the software package.

The examiner argues that it is not clear as to which "plurality of software packages" is to be associated with "a software package". 10/22/04 Office Action, p. 4, § 6.

Since claims 24 and 49 are limitations applicable to the second element of claims 1 and 26 (i.e. they deal with execution of the software packages), it is clear that they must be associated with the "plurality of software packages" that appears in the second element of both claims.

CLAIMS 2 AND 27

Claims 2 and 27 read as follows:

2. The method of claim 1 wherein the plurality of software packages of the "executing" step includes only valid software packages, the method further comprising the step:
utilizing one or more tests to identify the software packages that are valid.

27. The apparatus of claim 26 wherein the plurality of software packages executed by the "executing" means includes only valid software packages, the apparatus further comprising:
a means for utilizing one or more tests to identify the software packages that are valid.

The examiner states that "it is not clearly understood what 'valid' here mean, valid with respect to what, or what is the difference between valid and non-valid software package." [sic]
10/22/04 Office Action, p. 4, § 6.

A "valid" software package is a software package that survives the "validation" process, a term of art defined as follows:

"Two kinds of risks are addressed during a software test. The first risk is that the ultimate software product will fail to meet the needs and expectations of the end user. These expectations comprise the system requirements, and the process of determining the likely extent to which the eventual software product will satisfy its requirements is called *validation*." *Encyclopedia of Computer Science*, Third Edition, Ralston & Reilly, editors, Software Testing, p. 1246, Van Nostrand Reinhold, New York, N.Y., 1993.

A person skilled in the art of computer systems would be well aware of the meaning of "valid".

CLAIMS 3 AND 28

Claims 3 and 28 read as follows:

3. The method of claim 2 wherein one of the tests for validity is a one's complement checksum test of a software package's program memory.

28. The apparatus of claim 27 wherein one of the tests for validity is a one's complement checksum test of a software package's program memory.

The examiner wonders whether the validity test claimed should be expressed as a "checksum test of a software package" rather than a "checksum test of a software package's program memory".

10/22/04 Office Action, p. 4, § 6.

The software package has been checked for validity prior to being loaded into the software package's program memory in the computer system where it is to be executed. The question to be answered by the checksum test is whether the software package's program memory has successfully captured an error-free version of the software package and it is appropriate to refer to the test as being a "checksum test of a software package's program memory" rather than the clearly misleading description of the test as being a "checksum test of a software package".

CLAIMS 6-7 AND 31-32

Claims 6-7 and 31-32 read as follows:

6. The method of claim 2 wherein a software package is assigned its own dedicated memory region, the software package's dedicated memory region including a stack memory region and/or a heap memory region, one of the tests for validity being whether the stack memory range and/or the heap memory range assigned during the execution of the software package's initialization procedure and the various associated entry points lies within the software package's dedicated memory region.

7. The method of claim 6 wherein one of the tests is whether the stack memory range and/or the heap memory range and the various associated entry points are returned within a predetermined time.

31. The apparatus of claim 27 wherein a software package is assigned its own dedicated memory region, the software package's dedicated memory region including a stack memory region and/or a heap memory region, one of the tests for validity being whether the stack memory range and/or the heap memory range assigned during the execution of the software package's initialization procedure and the various associated entry points lies within the software package's dedicated memory region.

32. The apparatus of claim 31 wherein one of the tests is whether the stack memory range and/or the heap memory range and the various associated entry points are returned within a predetermined time.

The examiner argues that the use of "and/or" causes the claims to be unclear. 10/22/04 Office Action, p. 5, § 6.

The dictionary states that "and/or" is "used to imply that any or all of the things named may be affected: *insurance covering fire and/or wind damage*". *The Random House College Dictionary, Revised Edition*, Random House, Inc., New York, N.Y. (1988). In accordance with the dictionary-approved usage, "a stack memory region and/or a heap memory region" should be interpreted as meaning "a stack memory region or a heap memory region or both".

The examiner argues that the claim language is unclear "specially when a stack region and a heap region in a memory constitute different purpose and different function." [sic] 10/22/04 Office Action, p. 5, § 6.

The examiner's conclusion is exactly opposite to what it should be. The claims are particularly clear because a stack memory region and a heap memory region ARE different in purpose and function. There is no conflict between the stack memory range and the various associated entry points lying within or without the software package's dedicated memory region and the heap memory range and the various associated entry points lying within or without the software package's dedicated memory region.

CLAIMS 18 AND 43

Claims 18 and 43 read as follows:

18. The method of claim 1 wherein safety-critical software is placed in one or more separate partitions thereby isolating the safety-critical software from non-safety-critical software.

43. The apparatus of claim 26 wherein safety-critical software is placed in one or more separate partitions thereby isolating the safety-critical software from non-safety-critical software.

Clarity

The examiner argues that "it is not clearly understood what 'partitions' mean." 10/22/04 Office Action, p. 5, § 6.

"Partition" is a term of art in computer science which means "a reserved portion of a computer memory". *McGraw-Hill Dictionary of Scientific and Technical Terms, Fourth Edition*, McGraw-Hill< Inc., New York, N.Y. (1989).

A person skilled in the computer science art would have no difficulty in understanding the meaning of "partition".

Antecedent Basis

The examiner argues that the term "safety-critical software" in the first line of the claims and "non-safety-critical software" in the second line of the claims has no antecedent basis. 10/22/04 Office Action, p. 5, § 6.

"Safety-critical software" appears for the first time in line 1 and "non-safety-critical software" appears for the first time in line 2 of claims 18 and 43. There can be no problem in antecedent basis with a term that is appearing for the first time unless the term is preceded by "the" or "said". Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2173.05(e).

CLAIMS 22 AND 47

Claims 22 and 47 read as follows:

22. The method of claim 1 wherein the presence of those software packages that are present is detected.

47. The apparatus of claim 26 wherein the presence of those software packages that are present is detected.

Clarity

The examiner argues that "it is not clearly understood what 'presence' is referring to, and where is the location of these software packages to detected, and software packages are identified by 'those'." [sic] 10/22/04 Office Action, p. 5, § 6.

Claims 1 and 26 states that a "plurality of software packages" are to be executed utilizing a common set of computational resources. A person skilled in the art would recognize that an essential component of the "computational resources" is a memory where the "plurality of software packages" would be stored. The user of the method or apparatus presumably knows the identity of the software packages that he intends to execute. However, it would be prudent to have the computational resources provide confirmation by detecting "the presence of those software packages that are present." Surely, the language of claims 22 and 47 are clear statements of this task.

The examiner seems puzzled by the phrase "those software packages that are present". The need for referring to "those software packages that are present" is simply a reflection of the "plurality of software packages" referred to in the second elements of claims 1 and 26 possibly being a subset of the "plurality of software packages" referred to in the preambles of claims 1 and 26.

Antecedent Basis

The examiner argues that "the presence of those software packages that are present" has insufficient antecedent basis. 10/22/04 Office Action, p. 5, § 6

The term "those software packages that are present" is a newly-defined group of software packages (one or more of the "plurality of software packages" referred to in the second element of claims 1 and 26) and needs no antecedent basis. The words "the presence of" placed in front of "those software packages that are present" does not create an antecedent problem since "the presence of" merely states a property of "those software packages that are present", i.e. the property of "being present", which is the property to be detected by the computational resources.

There simply is no basis for requiring an antecedent basis for a phrase such as "the presence of those software packages that are present". *See* Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2173.05(e).

III. WHETHER CLAIM 25 IS UNPATENTABLE UNDER 35 U.S.C. § 112, SECOND PARAGRAPH, FOR FAILING TO SET FORTH THE SUBJECT MATTER WHICH APPLICANTS REGARD AS THEIR INVENTION.

Claim 25 reads as follows:

25. Apparatus for practicing the method of claim 1.

The examiner argues that claim 25 is indefinite based on *Ex parte Lyell*, 17 USPQ2d 1548 (BPAI 1990). Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2173.05(p)) because it "claims both an apparatus and the method steps of using the apparatus." 10/22/04 Office Action, pp. 5-6, § 7. But claim 25 does not claim "both an apparatus and the method steps of using the apparatus." Claim 25 claims an "apparatus for practicing the method of claim 1" but does not include method steps of USING the "apparatus for practicing the method of claim 1". *Ex parte Lyell* does not apply.

To illustrate how different the formats of appellants' claim 25 and *Ex parte Lyell* claim 2 are, we reproduce *Ex parte Lyell* claim 2 below:

2. An automatic transmission tool in the form of a workstand and method for using same comprising:

a support means,

and [sic] internally splined sleeve affixed upright to said support means,

a threaded adjustment bolt threadably engaged through a hole in the bottom of said support means and projecting upward through said support frame into said sleeve,

and further comprising the steps of

1. positioning the output end of an automatic transmission onto said upright sleeve,

2. removing the internal components of said automatic transmission from the casing of said transmission,
3. repairing and replacing said internal components back into said casing, and
4. adjusting said internal components for fit and interference by means of adjusting said upwardly projecting adjustment bolt.

Note the set of elements pertaining to the apparatus and the separate set of steps pertaining to using the apparatus. Appellants' claim 25 defines the apparatus by the method it practices. There are no steps in the claim that define how the apparatus might be used.

Claim 25 is not indefinite and does not violate 35 U.S.C. § 112, second paragraph.

**IV. WHETHER CLAIM 25 IS UNPATENTABLE UNDER 35 U.S.C. § 101
BECAUSE IT EMBRACES OR OVERLAPS TWO DIFFERENT
STATUTORY CLASSES.**

Claim 25 reads as follows:

25. Apparatus for practicing the method of claim 1.

The examiner also argues that claim 25 violates 35 U.S.C. § 101 because "the claim is directed to neither a 'process' nor 'machine', but rather embraces or overlaps two different statutory classes of invention." 10/22/04 Office Action, p. 6, § 9.

Even though the examiner considers the format of appellants' claim 25 to be a violation of 35 U.S.C. § 101, such a format is implicitly approved in the Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 806.05(e), third paragraph.

Moreover, a similar format ("a method which utilizes the nozzle of claim 7") was one of the issues of *Ex parte Porter*. And this claim also (in the words of the examiner) "embraces or overlaps two different statutory classes." The actual claim is shown below:

6. A method for unloading non-packed, non-bridging and packed, bridging flowable particle catalyst and bead material from the opened end of a reactor tube which comprises utilizing the nozzle of claim 7.

In considering the format of this claim, the Board of Patent Appeals and Interferences concluded:

Turning to the rejection of claim 6 under 35 USC Section 112, second and fourth paragraphs, we do not agree with the examiner that the claim is either ambiguous or non-statutory.

Ex parte Porter, 25 USPQ2d 1144, 1147 (BPAI 1992)

Claim 25 does not violate 35 U.S.C. § 101.

V. WHETHER CLAIMS 1, 8, 23, 25, 26, 33, AND 48 ARE UNPATENTABLE UNDER 35 U.S.C. § 102(B) AS BEING ANTICIPATED BY BLUM ET AL. (U.S. 4,109,311).

"A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros. v. Union Oil Co. of California*, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987). (Cited in Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2131.)

"The identical invention must be shown in as complete detail as is contained in the . . . claim." *Richardson v. Suzuki Motor Co.*, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). (Cited in Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2131.)

CLAIMS 1, 25, AND 26

Claim 1 reads as follows:

1. *A method [1] for repetitively executing a plurality of software packages at one or more rates, utilizing a common set of computational resources, the method comprising the steps:*
 - [2] generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages;*
 - [3] executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals.*

Apparatus claims 25 and 26 also contain limitations [1], [2], and [3].

The limitations shown in boldface are not disclosed by Blum et al.

Limitation [1]

Blum et al. does not disclose *repetitively executing a plurality of software packages at one or more rates*.

The rate at which a software package is repetitively executed is equal to the reciprocal of the repetition period. If the repetition period is not the same from one period to the next, one cannot speak of a software package having a "rate" of execution.

The concept of a software package "execution rate" is not to be found in Blum et al. In fact, Blum et al.'s disclosure indicates that there is no such thing as an "execution rate" for an application executed by the Blum et al. invention. Specifically, Blum et al. discloses how applications are executed in the following words:

"[T]here is provided a time slice control mechanism for controlling the sequence in which the instructions are supplied to the execution mechanism for causing the instructions from the different programs to be executed in an interleaved manner." Blum et al., col. 2, lines 20-24.

The Blum et al. invention provides a means for allocating differing amounts of computing time to the different programs being executed:

"If, for example, four programs are provided for a processing unit and if a string of sixteen index words is chosen, then each program or microprogram can occupy anywhere from one-sixteenth to thirteen-sixteenths of the total computing time." Blum et al., col. 5, lines 10-15.

For purposes of illustration, consider two programs A and B which are listed for execution in 8 time slices as follows:

. . . |ABABABAB|ABABABAB| . . .

The vertical separators denote the beginnings of time-slice cycles. It would appear at this point that each program is being executed at a rate of $1/2T$, T being the duration of a time slice and the execution of each program occurring at periodic intervals of $2T$.

However, the above execution sequence does not necessarily occur because of Blum et al.'s "instruction execution modification mechanism." Blum et al., Abstract. Blum et al. points out that variations in time-slice assignments may occur from one time-slice cycle to the next:

"If it so happens that one or two of the programs are not required for a short period of time, then the computing time can be allocated among the remaining programs. Also, the individual programs can AUTOMATICALLY add to or reduce their share of the computing time. When a program adds to its computing time, the additional time is taken from one or more of the other programs which, at that moment, are being executed under uncritical time conditions." Blum et al., col. 5, lines 15-23, emphasis added.

Thus, the Blum et al. execution of programs initially listed as ABABABAB might proceed in the following fashion:

. . . |AAABABAA|BBABABBB| . . .

where Program A has taken two extra time slices away from Program B in the first time-slice cycle, and Program B has taken two time slices away from Program A in the second time-slice cycle. Note that the periods between executions of Program A are T, T, 2T, 2T, T, 3T, and 2T. The periods between executions are no longer the same and one can no longer say that Program A is executed at a particular rate. Similarly, the periods between executions of Program B are 2T, 3T, T, 2T, 2T, T, and T. Here also, one can no longer say that Program B is executed at a particular rate.

The Blum et al. invention, because of its "instruction execution modification mechanism", does not have a program execution rate and consequently does not disclose the claim-1 preamble limitation.

It is sometimes argued that a limitation appearing in the preamble of a claim such as limitation [1] is merely a statement of the intended use of the device. However, the Manual of Patent Examining Procedure emphasizes:

"The claim preamble must be read in the context of the entire claim. The determination of whether preamble recitations are structural limitations or mere

statements of purpose or use 'can be resolved only on review of the entirety of the [record] to gain an understanding of what the inventors actually invented and intended to encompass by the claim.' *Corning Glass Works v. Sumitomo Elec. U.S.A., Inc.*, 868 F.2d 1251, 1257, 9 USPQ2d 1962, 1966 (Fed. Cir. 1989)." Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2111.02.

For example, the "for use" clause in "a bicycle for use in getting from one place to another" is merely a statement of use and does not imply any structural limitations on the bicycle. On the other hand, the "for use" clause in "a cell phone for use in the Verizon Wireless network" is not merely a statement of use but also implies certain structural limitations on the cell phone so that it has the capability of being used in the Verizon Wireless network.

In the case of appellants' "*a method [1] for repetitively executing a plurality of software packages at one or more rates*", the "for use" clause, like the cell phone example, is not merely a statement of use but also implies limitations to the functions performed in the steps of the method. In such cases, the Manual of Patent Examining Procedure states:

"Any terminology in the preamble that limits the structure of the claimed invention must be treated as a claim limitation. See, e.g., *Corning Glass Works v. Sumitomo Elec. U.S.A., Inc.*, 868 F.2d 1251, 1257, 9 USPQ2d 1962, 1966 (Fed. Cir. 1989); *Pac-Tec Inc. v. Amerace Corp.*, 903 F.2d 796, 801, 14 USPQ2d 1871, 1876 (Fed. Cir. 1990). See also *In re Stencel*, 828 F.2d 751, 4 USPQ2d 1071 (Fed. Cir. 1987)" Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2111.02.

One might ask whether appellants' "*a method [1] for repetitively executing a plurality of software packages at one or more rates*" asserts limitations that are not already fully and intrinsically set forth by the limitations in the body of the claim:

"If the body of a claim fully and intrinsically sets forth all of the limitations of the claimed invention, and the preamble merely states, for example, the purpose or intended use of the invention, rather than any distinct definition of any of the claimed invention's limitations, then the preamble is not considered a limitation and is of no significance to claim construction. *Pitney Bowes, Inc. v. Hewlett-Packard Co.*, 182 F.3d

1298, 1305, 51 USPQ2d 1161, 1165 (Fed. Cir. 1999)." **Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2111.02.**

Neither of the body elements of the claim at issue disclose either separately or in combination "*a method [1] for repetitively executing a plurality of software packages at one or more rates*", and thus, these words represent an additional structural limitation which must be treated as a legitimate claim limitation and not merely "a statement of purpose or use."

The examiner argues that limitation [1] is disclosed by Blum et al.'s disclosure assigning different programs different numbers of time slices during a time-slice cycle. 10/22/04 Office Action, p. 6, § 12. However, assigning different numbers of time slices to different programs is not the same as repetitively executing different programs at the same or different rates. As we have pointed out above, there is no rate of execution that can be established for any of Blum et al.'s programs, regardless of how many time slices are assigned to a program.

Limitation [2]

Blum et al. does not disclose *generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages*.

Blum et al. discloses a system wherein the programs to be executed are listed one or more times on a program list. The programs on the list are executed repetitively in the order they appear on the list. Each entry on the program list is allotted one time slice interval for execution. Since a program may be listed more than once on the program list, its total allotment may amount to a number of time slice intervals during the execution of the program list. Blum et al., col. 5, lines 4-23.

Blum et al. generates a periodic sequence of pulses which delineate the time slices and time the execution process. A pulse denotes the end of one time slice and the beginning of the next. The occurrence of a pulse results in the selection for execution of the next item on the program list after the currently-executing program. Blum et al., col. 6, line 64 - col. 7, line 10.

The pulse sequence generated in the Blum et al. invention (see Fig. 7, "clock pulse T") does not correspond to Limitation [2] in that Blum et al.'s pulse sequence is a single sequence that is not assigned to any program. Each of Blum et al.'s pulses merely marks the end of one time slice interval and the beginning of the next. Limitation [2] claims a plurality of sequences, each sequence being assigned to each of a plurality of software packages. In addition, each time interval in one of appellants' sequences is the time interval when the associated software package executes (see Limitation [3])

The examiner argues that Blum et al. discloses *generating a sequence of time intervals for each of the plurality of software packages* in the following passages (10/22/04 Office Action, p. 7, § 12):

"The time slices in any given processing unit can be allocated either rigidly, that is, the machine time available is divided equally among the different programs, or dynamically, that is, the program with the highest priority or the program requiring the greatest amount of processing time is allocated a greater number of time slices at the expense of low priority programs or programs requiring less processing time." Blum et al., col. 3, lines 47-54.

"Time slicing is accomplished by means of a time slice control unit 37 which supplies to the execution unit 36 a program pointer (PGM PTR) and an access pointer (ACC PTR). A primary difference between these pointers is that the program pointer points to the program for the currently executing time slice and the access pointer points to the program for the next following time slice. Blum et al., col. 6, lines 2-9

There is nothing in these passages that relates to *generating a sequence of time intervals*.

The examiner argues that Blum et al. discloses *the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages* in the following passage (10/22/04 Office Action, p. 7, § 12):

"With this type of time slice control, each program in a given processing unit is sequentially allocated time slices for the duration of which all the resources of the processing unit are made available to one program. For the duration of the subsequent time slice, the processing unit is exclusively available to the next program in the sequence." Blum et al., col. 3, lines 40-46.

The phrase *the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages* further defines *generating a sequence of time intervals for each of the plurality of software packages*. The passage cited by the examiner has nothing to do with further specifying the nature of the sequences of time intervals that are the subject of Limitation [2].

There is no disclosure of Limitation [2] in Blum et al.

Although Limitation [2], reasonably interpreted, is not disclosed by Blum et al., it would seem only prudent for appellants to argue the non-equivalency of the Blum et al. invention with respect to appellants' embodiment as described in the specification and drawings pursuant to 35 U.S.C. 112, ¶ 6. *See In re Donaldson Co.*, 16 F.3d 1189, 29 USPQ2d 1845 (Fed. Cir. 1994).

The statute applies inasmuch as Limitation [2] is simply a statement of a function to be performed without the recitation of acts for performing that function. Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2181 I. Limitation [2] speaks only of *generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages*.

Appellants' specification describes a particular embodiment of apparatus for performing the function of Limitation [2]:

"An expanded version of the invention is shown in Fig. 3. The positive transitions of a clock signal are counted in a five-bit counter. The counter values repeat after every 32 clock transitions. Time is divided into slots that are assigned numbers in accordance with the three most significant bits of the counter value. Each slot is divided into four sub-slots that are assigned numbers in accordance with the two least significant bits of the counter value." Appellants' Specification, p.6, lines 3-7.

The counter values shown in Fig. 3 identify a sequence of time intervals available for allocation to the software packages to be executed. For example, a software package to be executed at Rate 1 could be assigned the sequence of time intervals marked with X's in the Rate 1 column. Similarly, software packages to be executed at Rate 2 and Rate 3 could be assigned the sequences of time intervals marked with X's in the Rate 2 and Rate 3 columns. This, of course, is only one of many possible ways of assigning time intervals to software packages. The result, obviously, is the accomplishment of the function *generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages.*

There are no acts nor apparatus described in Blum et al. that are equivalent to those employed by appellants' in performing the function of Limitation [2]. Blum et al.'s clock 52 (Fig. 50) appears to have only two functions. It supplies a set/reset input of latch circuit 66 (Fig. 6) (col. 8, lines 53-55) and activates the transfer of the index word pointer in APSAR 35a to PPSAR 48 (Fig. 5) (col. 7, lines 3-7). Neither of these actions are analogous to performing the sequence-generating function of Limitation [2].

There are no acts or apparatus described in Blum et al. that are the equivalent of the acts and apparatus described in appellants' specification and drawings for performing the Limitation [2] function.

Limitation [3]

Blum et al. does not disclose *executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals.*

Blum et al. does not disclose Limitation [2] (generating a sequence of time intervals for each software package) and obviously could not disclose Limitation [3] which requires such sequences for its execution.

To disclose this limitation, Blum et al.'s apparatus would have to (1) identify the next time interval, (2) identify the sequence of time intervals of which the next time interval is a member, (3) identify the program associated with the identified sequence of time intervals, and (4) execute the identified program beginning with the start of the next time interval. Rather than performing this process, Blum et al.'s index word buffer 35 begins execution of the next program on the list of programs to be executed whenever a regularly-occurring clock pulse T arrives. Blum et al., col. 7, lines 3-10. Blum et al.'s approach to selecting a program for execution has nothing to do with selecting an program based on its association with the sequence of time intervals of which the upcoming time interval is a member.

The examiner argues (10/22/04 Office Action, p. 7, § 12) that Limitation [3] is disclosed by the following passage from Blum et al.:

"An instruction execution modification mechanism . . . wherein multiple programs or tasks are performed in a concurrent manner by means of a time slice mechanism which causes the instructions from the different programs to be executed in an interleaved manner. Instructions from the different programs are executed during different successive time slice intervals." Blum et al., Abstract, lines 1-8.

This passage says nothing about *each software package being executed during the time intervals of its sequence of time intervals* that were allocated as a result of performing the function of Limitation [2].

Here also, it is appropriate to invoke 35 U.S.C. 112, ¶ 6. *See In re Donaldson Co.*, 16 F.3d 1189, 29 USPQ2d 1845 (Fed. Cir. 1994). Limitation [3] qualifies in that Limitation [3] is simply a statement of a function to be performed without the recitation of acts for performing that function. Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2181 I. Limitation [3] speaks only of *executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals*.

It follows from the words of Limitation [3] and further supported by Appellants' specification and drawings that the steps involved in performing the function of Limitation [3] are: (1) identify the next time interval, (2) identify the sequence of time intervals of which the next time interval is a member, (3) identify the program associated with the identified sequence of time intervals, and (4) execute the identified program beginning with the start of the next time interval. Appellants' Specification, p. 6, line 8 - p. 7, line 17.

There are no acts nor apparatus described in Blum et al. that are equivalent to those employed by appellants' in performing the function of Limitation [3]. Blum et al.'s apparatus simply executes the program next on the list after the program currently being executed. There is no equivalency between appellants' embodiment for executing programs and the apparatus described in Blum et al.

* * * * *

Blum et al. does not disclose any of the boldface limitations of claim 1 and consequently did not anticipate applicants' claim-1 invention.

Claim 25, which depends from claim 1, was therefore also not anticipated by Blum et al.

Claim 26 is a claim for apparatus that includes all of the limitations of method claim 1. Consequently, as in the case of appellants' claim-1 invention, appellants' claim-26 invention also was not anticipated by Blum et al.

CLAIMS 8 AND 33

Claim 8 reads as follows:

8. *The method of claim 1 wherein a software package is assigned its own dedicated memory region.*

Apparatus claim 33 contains the same limitation as claim 8.

The dictionary definition of "dedicated" is "designed, used or reserved for a specific purpose." *The Random House College Dictionary Revised Edition*, Random House, Inc, New York, N.Y., 1988.

Blum et al. does not disclose a program being assigned its own dedicated memory region.

The examiner cites Blum et al., col. 3, line 65, through col. 4, line 13, and col. 4, lines 50-54 as disclosing the limitation of claims 8 and 33 (10/22/04 Office Action, p. 7, § 15), but it does not. The first passage cited states that "the processing unit 13 includes a control storage 26" (col. 3, lines 66-67) and that a "plurality of control programs or microprograms . . . are stored in the control storage 26 (col. 4, lines 4-7)." Nothing is said about each program having its own dedicated memory

region. The second passage cited by the examiner discusses the operation of a plural-bit pointer (PTR). Here also, nothing is said about each program having its own dedicated memory region.

Blum et al. does not disclose limitations of claims 8 and 33 and consequently did not anticipate appellants' claim-8 and claim 33 inventions.

CLAIMS 23 AND 48

Claim 23 reads as follows:

23. *The method of claim 1 wherein one or more of the plurality of software packages are independently compiled, linked, and loaded.*

Apparatus claim 48 contains the same limitation as claim 23.

Nothing is said in Blum et al. as to how the programs are compiled, linked, and loaded.

The examiner argues that the limitations of claims 23 and 48 are INHERENTLY disclosed by the same passages cited above in connection with claims 8 and 33. 10/22/04 Office Action, p. 8, § 16. These passages discuss the storage and readout of programs and the operation of a plural-bit pointer. There is no disclosure relating to compiling, linking, and loading programs, inherent or otherwise.

**VI. WHETHER CLAIMS 10-11, 13-14, 18, 35-36, 38-39, AND 43 ARE
UNPATENTABLE UNDER 35 U.S.C. § 103(A) IN VIEW OF BLUM ET AL.
(U.S. 4,109,311).**

Three basic criteria must be satisfied to establish *prima facie* obviousness:

"The legal concept of *prima facie* obviousness is a procedural tool of examination which applies broadly to all arts. . . . The examiner bears the initial burden of factually supporting any *prima facie* conclusion of obviousness. If the examiner does not produce a *prima facie* case, the applicant is under no obligation to submit evidence of nonobviousness. . . . To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed. Cir. 1991)." Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2142.

CLAIMS 10 AND 35

Claim 10 reads as follows:

10. *The method of claim 1 wherein a software package includes background tasks as well as foreground tasks, the background tasks being performed after the foreground tasks have been completed.*

Apparatus claim 35 contains the same limitation as claim 10.

Blum et al. does not disclose anything relating to the performance of background and foreground tasks in executing a program. The examiner argues that it would have been obvious to a person skilled in the art to divide the tasks performed by one of Blum et al.'s programs into foreground and background tasks and causing the background tasks to be executed after the foreground tasks have been completed. In making this assertion, the examiner seems to ignore the

complexities of the Blum et al. system in managing the interleaved execution of a multitude of programs and accomplishing both dynamic time slicing and dynamic instruction execution modification or function modification. Blum et al., col. 5, lines 52-65. Before concluding that it would be obvious to further complicate the Blum et al. system by adding foreground/background task segregation and execution, it is worth noting what the experts say:

"On a small computer, such as is used for process control, it is common to provide a *background/foreground* system that permits two programs to execute. . . . The two programs might cooperate by mutual understanding; i.e. the programmers could insure that each would not interfere with the use of the system by the other.

"While in very simple situations it may be feasible to multiprogram cooperatively, often we must be sure that, if one program somehow violates the rules of the system it does not corrupt the whole environment. Thus, most multiprogramming systems require a *monitor* (also called an *executive* or *supervisor*).

"It is the responsibility of the monitor to maintain the integrity of the system. In the case of the background/foreground system used as an example, we would like the monitor to guarantee that the foreground real-time program will be able to take its measurements, even if the background program goes into a loop and never voluntarily relinquishes control."

***Encyclopedia of Computer Science, Third Edition, Multiprogramming*, p. 908, Edited by Ralston & Reilly, IEEE Press, Van Nostrand Reinhold, New York, N.Y., 1993.**

As the encyclopedia article points out, it is common knowledge in process control to separate background and foreground tasks into two coordinated programs that are executed one before the other. And it would be proper in such situations to take official notice of such common knowledge in building a case of obviousness. However, it is not common knowledge that background/foreground execution concepts are readily applied to complex multitasking systems such as the one described in Blum et al. It certainly cannot be said that the applicability of background/foreground execution concepts to multitasking systems is "capable of instant and

unquestionable demonstration as being well-known." See Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2144.03 A. Quoting from the MPEP:

"Ordinarily, there must be some form of evidence in the record to support an assertion of common knowledge. See *Lee*, 277 F.3d at 1344-45, 61 USPQ2d at 1434-35 (Fed. Cir. 2002; *Zurko*, 258 F.3d at 1386, 59 USPQ2d at 1697 (holding that general conclusions concerning what is 'basic knowledge' or 'common sense' to one of ordinary skill in the art without specific factual findings and some concrete evidence in the record to support these findings will not support an obviousness rejection).

Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2144.03 B.

As was indicated at the beginning of this discussion of Issue VI, there are three criteria to be satisfied in establishing *prima facie* obviousness: (1) a teaching of the claim limitation, (2) motivation for combining reference teachings, and (3) reasonable expectation of success. In his 10/22/04 Office Action, p. 8, § 19, the examiner sought to provide the teaching of the claim limitation by asserting that foreground/background execution concepts are well known in the art. Insofar as motivation and expectation of success were concerned, the examiner had nothing to say.

The examiner has not established *prima facie* obviousness of claims 10 and 35.

CLAIMS 11 AND 36

Claim 11 reads as follows:

11. *The method of claim 10 wherein a background task is an infinite loop.*

Apparatus claim 36 contains the same limitation as claim 11.

The examiner attempted to establish *prima facie* obviousness of claims 11 and 36 by appending the following phrase to his argument for the obviousness of claims 10 and 35: "a task such as an infinite loop may be running in the background." 10/22/04 Office Action, p. 8, § 19.

In his argument for the obviousness of claims 11 and 36, the examiner does not even assert that it is well known in the art that a background task may be an infinite loop.

The examiner has not addressed any of the three criteria for establishing the *prima facie* obviousness of claims 11 and 36 and consequently, has not established the *prima facie* obviousness of claims 11 and 36.

CLAIMS 13 AND 38

Claim 13 reads as follows:

13. *The method of claim 1 wherein a failure in the execution of a software package causes information to be logged in a failure log.*

Apparatus claim 38 contains the same limitation as claim 13.

Blum et al. does not disclose anything relating to maintaining a failure log. The examiner argues that "it is well known in the art of program execution . . . to log execution failures linked to software packages causing the failure in order to record different occurrences, reasons, and locations of execution failures for future modifications." 10/22/04 Office Action, p. 8, § 20. However, maintaining a failure log does not appear to be an activity that is "capable of instant and unquestionable demonstration as being well-known" (*See Manual of Patent Examining Procedure*, 8th Edition, May 2004 Revision, § 2144.03 A) and thereby provide support for an obviousness conclusion. Quoting from the MPEP:

"Ordinarily, there must be some form of evidence in the record to support an assertion of common knowledge. See *Lee*, 277 F.3d at 1344-45, 61 USPQ2d at 1434-35 (Fed. Cir. 2002; *Zurko*, 258 F.3d at 1386, 59 USPQ2d at 1697 (holding that general conclusions concerning what is 'basic knowledge' or 'common sense' to one of ordinary skill in the

art without specific factual findings and some concrete evidence in the record to support these findings will not support an obviousness rejection).

Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2144.03 B.

If one consults the preeminent encyclopedia of computer science (*Encyclopedia of Computer Science, Third Edition*, Edited by Ralston & Reilly, IEEE Press, Van Nostrand Reinhold, New York, N.Y., 1993), one finds nothing relating to maintaining a failure log.

The experts on software reliability at Duke University make the following statement:

"Outages in computer systems consist of both hardware and software failures. While hardware failures have been studied extensively and varied mechanisms have been presented to increase the system availability with regard to such failures, software failures and the corresponding reliability/availability analysis has not drawn much attention from researchers. The study of software failures has now become more important since it has been recognized that computer systems outages are more due to software faults than to hardware faults."

Web Site: http://srel.ee.duke.edu/sw_ft/node1.html (Introduction).

If maintaining a failure log were a well known technique (i.e. capable of instant and unquestionable demonstration as being well-known) for achieving computer reliability and availability, surely the Duke paper referenced above would have something to say about the technique. And yet nothing is said in this paper about maintaining a failure log.

As was indicated at the beginning of this discussion of Issue VI, there are three criteria to be satisfied in establishing *prima facie* obviousness: (1) a teaching of the claim limitation, (2) motivation for combining reference teachings, and (3) reasonable expectation of success. In his 10/22/04 Office Action (p. 8, § 19), the examiner sought to provide the teaching of the claim limitation by asserting that maintaining a failure log is well known in the art. But case law states **"that general conclusions concerning what is 'basic knowledge' or 'common sense' to one of ordinary skill in the art without specific factual findings and some concrete evidence in the**

record to support these findings will not support an obviousness rejection." Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2144.03 B.

Insofar as motivation is concerned, the examiner opines that a failure log would be useful for future modifications even though the primary reference Blum et al. provides no support for this viewpoint.

And insofar as expectation of success is concerned, the examiner had nothing to say.

The examiner has not established *prima facie* obviousness of claims 13 and 38.

CLAIMS 14 AND 39

Claim 14 reads as follows:

14. *The method of claim 13 wherein a failure in execution is linked to the software package that caused the failure.*

Apparatus claim 39 contains the same limitation as claim 14.

The examiner takes official notice that it is well known to link a failure in execution to the software package that caused the failure and then argues that it would be obvious to incorporate such a capability in the invention of Blum et al. 10/22/04 Office Action, p. 8, § 20.

Linking a failure in execution to the software package that caused the failure is not something that is "capable of instant and unquestionable demonstration as being well-known." See Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2144.03 A. A "software monitor" is the software instrument that measures performance of a computer program and is the instrument that would be used in obtaining and analyzing failures in performance of computer programs. The "Software Monitor" article in a preeminent computer science encyclopedia had nothing to say about

linking failures in execution to the software package that caused the failure. *Encyclopedia of Computer Science, Third Edition*, Software Monitor, pp. 1238-1240, Edited by Ralston & Reilly, IEEE Press, Van Nostrand Reinhold, New York, N.Y., 1993)

Quoting from the MPEP:

"Ordinarily, there must be some form of evidence in the record to support an assertion of common knowledge. See *Lee*, 277 F.3d at 1344-45, 61 USPQ2d at 1434-35 (Fed. Cir. 2002; *Zurko*, 258 F.3d at 1386, 59 USPQ2d at 1697 (holding that general conclusions concerning what is 'basic knowledge' or 'common sense' to one of ordinary skill in the art without specific factual findings and some concrete evidence in the record to support these findings will not support an obviousness rejection).

Manual of Patent Examining Procedure, 8th Edition, May 2004 Revision, § 2144.03 B.

The examiner essentially argues that a person skilled in the art, after reading and digesting Blum et al., would conclude that what the Blum et al. system needs is a way of identifying when failures in execution of programs occur and to link these failures to the programs that caused them. But there is nothing in Blum et al. which would suggest such a modification.

Blum et al. is wholly concerned with obtaining a multiprogramming system "whereby the time slice assignments can be dynamically changed to best meet the needs arising at any particular moment . . . [and] the function modification action operates to further advantage since a smaller amount of microinstruction code and hence a smaller control storage are required and the microprogramming is simpler and more flexible. Blum et al., col. 12, lines 27-35. Blum et al. is seeking simpler and more flexible microprogramming and a smaller amount of microinstruction code and the examiner suggests that statements such as this would cause a person skilled in the art to add failure analysis and reporting thereby going in the direction of more complex and less flexible microprogramming and a greater amount of microinstruction code.

Moreover, the examiner finds motivation for a person skilled in the art to make such a change in the Blum et al. invention because it would provide a basis for future modifications even though there is not a hint in Blum et al. that such failures in execution should be of concern. The examiner does not discuss expectation of success in incorporating failure analysis and reporting in the Blum et al. invention.

The examiner has not established a case of *prima facie* obviousness of claims 14 and 39.

CLAIMS 18 AND 43

Claim 18 reads as follows:

18. *The method of claim 1 wherein safety-critical software is placed in one or more separate partitions thereby isolating the safety-critical software from non-safety-critical software.*

Apparatus claim 43 contains the same limitation as claim 18.

The examiner makes the statement that "Blum discloses software packages are assigned own memory partitions (col. 3, line 65 to col. 4, line 13; and col. 4, lines 50-54), but does not necessarily teach safety-critical and non-safety-critical software." 10/22/04 Office Action, p. 9, § 21.

The passages cited by the examiner use the term "storage locations" rather than "partitions". A "partition" is a term of art in computer science which means "one of a number of fixed portions into which a computer memory is divided in certain multiprogramming systems." *McGraw-Hill Dictionary of Scientific and Technical Terms, Fourth Edition*, McGraw-Hill, Inc., New York, N.Y. (1989). Blum et al. does not disclose anything relating to "safety-critical" or "non-safety-critical software" nor does Blum et al say anything about utilizing "partitions" for the storage of programs.

The examiner has not cited any prior art which discloses the use of memory partitions for the storage of safety-critical programs in multiprogramming computer systems. Other than making a conclusory statement that "it would have been obvious to one skilled in the art at the time of the invention to place the safety-critical and the non-safety critical software in separate partitions" (10/22/04 Office Action, , p. 9, § 21), the examiner has not provided any rationale as to why a person skilled in the art would be motivated to modify the Blum et al. invention so that safety-critical programs are placed in one or more separate partitions away from non-safety-critical programs. And the examiner does not even discuss the expectation of success in achieving such a modification.

Thus, the examiner has not shown that any of the three criteria for obviousness has been met and has not established *prima facie* obviousness of claims 18 and 43.

CONCLUSIONS

Appellants' invention is a method and apparatus for repetitively executing a plurality of software packages at a plurality of rates utilizing a common set of computational resources and predictive scheduling. Predictive scheduling is accomplished by generating a sequence of time intervals for each of the plurality of software packages and executing each of the plurality of software packages during each time interval in its sequence of time intervals.

The examiner rejected all of appellants' claims. The cornerstone of the examiner's rejections is the Blum et al. patent which discloses a just-in-time scheduling technique whereby a list of software packages is maintained, and the software packages on the list are executed in sequence. When the time allotted to the execution of one software package ends, the execution of the next software package on the list begins. There is no way of predicting when any of the software packages on the list will begin execution in the future.

A key step in appellants' predictive scheduling method is "generating a sequence of time intervals for each of the plurality of software packages." The examiner argues that this step is disclosed by Blum et al. as follows:

"The time slices in any given processing unit can be allocated either rigidly, that is, the machine time available is divided equally among the different programs, or dynamically, that is, the program with the highest priority or the program requiring the greatest amount of processing time is allocated a greater number of time slices at the expense of low priority programs or programs requiring less processing time." Blum et al., col. 3, lines 47-54.

"Time slicing is accomplished by means of a time slice control unit 37 which supplies to the execution unit 36 a program pointer (PGM PTR) and an access pointer (ACC PTR). A primary difference between these pointers is that the program pointer points to the program for the currently executing time slice and the access pointer points to the program for the next following time slice. Blum et al., col. 6, lines 2-9

Clearly, the examiner's contention is incorrect. There is nothing in the Blum et al. passages quoted above which even imply the step of "generating a sequence of time intervals for each of the plurality of software packages." In fact, there is nothing in Blum et al. or any of the other references cited by the examiner which disclose the limitations of any of the 49 claims which have been rejected by the examiner.

The Board should reverse the examiner's rejection of the 49 claims subject to appeal.

CLAIMS APPENDIX

1. A method for repetitively executing a plurality of software packages at one or more rates, utilizing a common set of computational resources, the method comprising the steps:

generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages;

executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals.

2. The method of claim 1 wherein the plurality of software packages of the "executing" step includes only valid software packages, the method further comprising the step:

utilizing one or more tests to identify the software packages that are valid.

3. The method of claim 2 wherein one of the tests for validity is a one's complement checksum test of a software package's program memory.

4. The method of claim 2 wherein a software package is assigned its own dedicated memory region, one of the tests for validity being whether the address returned for the software package's initialization procedure lies within its dedicated memory region.

5. The method of claim 4 wherein one of the tests is whether the address is returned within a predetermined time.

6. The method of claim 2 wherein a software package is assigned its own dedicated memory region, the software package's dedicated memory region including a stack memory region and/or a heap memory region, one of the tests for validity being whether the stack memory range and/or the heap memory range assigned during the execution of the software package's initialization

procedure and the various associated entry points lies within the software package's dedicated memory region.

7. The method of claim 6 wherein one of the tests is whether the stack memory range and/or the heap memory range and the various associated entry points are returned within a predetermined time.

8. The method of claim 1 wherein a software package is assigned its own dedicated memory region.

9. The method of claim 8 wherein the software package's dedicated memory region includes a stack memory region, a software package's stack residing in the software package's stack memory region.

10. The method of claim 1 wherein a software package includes background tasks as well as foreground tasks, the background tasks being performed after the foreground tasks have been completed.

11. The method of claim 10 wherein a background task is an infinite loop.

12. The method of claim 10 wherein the software package causes the power utilized in executing the software package to be minimized after completion of the background tasks.

13. The method of claim 1 wherein a failure in the execution of a software package causes information to be logged in a failure log.

14. The method of claim 13 wherein a failure in execution is linked to the software package that caused the failure.

15. The method of claim 13 wherein quality of performance in executing a software package is represented by one or more performance-quality parameters, values of the one or more performance-quality parameters being determined from the information logged in a failure log, the

execution of a software package being subject to a plurality of execution options, an execution option being selected on the basis of one or more performance-quality parameter values.

16. The method of claim 15 wherein the plurality of execution options are user configurable.

17. The method of claim 15 wherein performance-quality parameters include the number of failures and/or the rate of failures for one or more classes of failures recorded in a software package's failure log.

18. The method of claim 1 wherein safety-critical software is placed in one or more separate partitions thereby isolating the safety-critical software from non-safety-critical software.

19. The method of claim 1 wherein each of the plurality of software packages is assigned its own memory block, a software package being enableable to read data only from zero or more memory blocks associated with other software packages, the zero or more memory blocks readable by a software package being either predetermined or determined during execution of the software packages in accordance with a set of one or more rules.

20. The method of claim 1 wherein each of the plurality of software packages is assigned its own memory block, a software package being enableable to write data only to zero or more memory blocks associated with other software packages, the zero or more memory blocks writeable by a software package being either predetermined or determined during execution of the software packages in accordance with a set of one or more rules.

21. The method of claim 1 wherein an executive software package enforces the discipline that each software package executes only during the time intervals of its sequence of time intervals, the executive software package determining when the execution of a software package

extends into a time interval belonging to the sequence of time intervals assigned to another software package and performs a remedial action.

22. The method of claim 1 wherein the presence of those software packages that are present is detected.

23. The method of claim 1 wherein one or more of the plurality of software packages are independently compiled, linked, and loaded.

24. The method of claim 1 wherein a software package has its own stack, the software package's stack being selected prior to executing the software package.

25. Apparatus for practicing the method of claim 1.

26. Apparatus for repetitively executing a plurality of software packages at a plurality of rates, the apparatus comprising:

a means for generating a sequence of time intervals for each of the plurality of software packages, the time intervals belonging to one software package not overlapping the time intervals belonging to any other of the plurality of software packages;

a means for executing a plurality of software packages, each software package being executed during the time intervals of its sequence of time intervals.

27. The apparatus of claim 26 wherein the plurality of software packages executed by the "executing" means includes only valid software packages, the apparatus further comprising:

a means for utilizing one or more tests to identify the software packages that are valid.

28. The apparatus of claim 27 wherein one of the tests for validity is a one's complement checksum test of a software package's program memory.

29. The apparatus of claim 27 wherein a software package is assigned its own dedicated memory region, one of the tests for validity being whether the address returned for the software package's initialization procedure lies within its dedicated memory region.

30. The apparatus of claim 29 wherein one of the tests is whether the address is returned within a predetermined time.

31. The apparatus of claim 27 wherein a software package is assigned its own dedicated memory region, the software package's dedicated memory region including a stack memory region and/or a heap memory region, one of the tests for validity being whether the stack memory range and/or the heap memory range assigned during the execution of the software package's initialization procedure and the various associated entry points lies within the software package's dedicated memory region.

32. The apparatus of claim 31 wherein one of the tests is whether the stack memory range and/or the heap memory range and the various associated entry points are returned within a predetermined time.

33. The apparatus of claim 26 wherein a software package is assigned its own dedicated memory region.

34. The apparatus of claim 33 wherein the software package's dedicated memory region includes a stack memory region, a software package's stack residing in the software package's stack memory region.

35. The apparatus of claim 26 wherein a software package includes background tasks as well as foreground tasks, the background tasks being performed after the foreground tasks have been completed.

36. The apparatus of claim 35 wherein a background task is an infinite loop.

37. The apparatus of claim 35 wherein the software package causes the power utilized in executing the software package to be minimized after completion of the background tasks.

38. The apparatus of claim 26 wherein a failure in the execution of a software package causes information to be logged in a failure log.

39. The apparatus of claim 38 wherein a failure in execution is linked to the software package that caused the failure.

40. The apparatus of claim 38 wherein quality of performance in executing a software package is represented by one or more performance-quality parameters, values of the one or more performance-quality parameters being determined from the information logged in a failure log, the execution of a software package being subject to a plurality of execution options, an execution option being selected on the basis of one or more performance-quality parameter values.

41. The apparatus of claim 40 wherein the plurality of execution options are user configurable.

42. The apparatus of claim 40 wherein performance-quality parameters include the number of failures and/or the rate of failures for one or more classes of failures recorded in a software package's failure log.

43. The apparatus of claim 26 wherein safety-critical software is placed in one or more separate partitions thereby isolating the safety-critical software from non-safety-critical software.

44. The apparatus of claim 26 wherein each of the plurality of software packages is assigned its own memory block, a software package being enableable to read data only from zero or more memory blocks associated with other software packages, the zero or more memory blocks readable by a software package being either predetermined or determined during execution of the software packages in accordance with a set of one or more rules.

45. The apparatus of claim 26 wherein each of the plurality of software packages is assigned its own memory block, a software package being enableable to write data only to zero or more memory blocks associated with other software packages, the zero or more memory blocks writeable by a software package being either predetermined or determined during execution of the software packages in accordance with a set of one or more rules.

46. The apparatus of claim 26 wherein an executive software package enforces the discipline that each software package executes only during the time intervals of its sequence of time intervals, the executive software package determining when the execution of a software package extends into a time interval belonging to the sequence of time intervals assigned to another software package and performs a remedial action.

47. The apparatus of claim 26 wherein the presence of those software packages that are present is detected.

48. The apparatus of claim 26 wherein one or more of the plurality of software packages are independently compiled, linked, and loaded.

49. The apparatus of claim 26 wherein a software package has its own stack, the software package's stack being selected prior to executing the software package.

EVIDENCE APPENDIX

Application Number: 09/821,537

P573C

Art Unit: 2154

COPY OF TERMINAL DISCLAIMER